Objectives

- Divide and Conquer Algorithms

Review

- What was the problem we were trying to solve on Monday?
  - What was our optimization goal?
- What was our solution to the problem?

DIVIDE AND CONQUER ALGORITHMS

Divide-and-Conquer

Divide et impera.
Veni, vidi, vici.
- Julius Caesar

- Divide-and-conquer process
  - **Break up** problem into **several parts**
  - Solve each part **recursively**
  - **Combine** solutions to sub-problems into overall solution
- Most common usage:
  - Break up problem of size n into two equal parts of size ½n
  - Solve two parts recursively
  - Combine two solutions into overall solution

Discussion

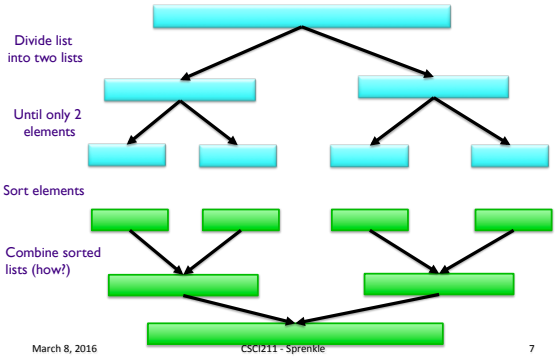
- What is a well-known divide and conquer algorithm?

Merge Sort

Merge Sort

- How does Merge Sort work?
- When do we stop?

Merge Sort



Merge Sort

```
MergeSort( L[1…n] ):
     if len(L) == 1:
          return L
     if len(L) == 2:
          compare the two entries in L,
          swap if necessary
          return L
     A = MergeSort(L[:n/2])
     B = MergeSort(L[n/2+1:])
     M = Merge(A, B)
     return M
```


RECURRENCE RELATIONS

Analyzing Merge Sort

General Template
- Break up problem of size n into two equal parts of size ½n
- Solve two parts recursively
- Combine two solutions into overall solution

- **Def.** T(n) = number of comparisons to mergesort an input of size *n*
- Want to say a bit more about what T(n) is
  - Break it down more…

What can we say about the running time w.r.t. to the different parts of the above template?

Analyzing Merge Sort

General Template
- Break up problem of size n into two equal parts of size ½n — **O(1)**
- Solve two parts recursively — **T(n/2) + T(n/2)**
- Combine two solutions into overall solution — **O(n)**

- **Def.** T(n) = number of comparisons to mergesort an input of size *n*
- Want to say a bit more about what T(n) is
  - Break it down more…

What is the base case? Its running time?

Merge Sort's Recurrence Relation

- Put an *upperbound* on T(n):

O(n)

For some constant c,
$T(n) \leq 2\,T(n/2) + cn$ when $n > 2$,
$T(2) \leq c$

Solve T(n) to come up with explicit bound

Merge Sort's Recurrence Relation

```
MergeSort( L[1…n] ):
     if len(L) == 1:
          return L          Base cases
     if len(L) == 2:
          compare the two entries in L,
          swap if necessary
          return L
     A = MergeSort(L[:n/2])    T(n/2)
     B = MergeSort(L[n/2+1:]) T(n/2)
     M = Merge(A, B)           O(n)
     return M
```

T(n) = 2T(n/2) + O(n)

Approaches to Solving Recurrences

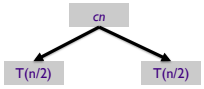
1. Unroll recursion
   - Look for patterns in runtime at each level
   - Sum up running times over all levels
2. Substitute guess solution into recurrence
   - Check that it works
   - Induction on n

Unrolling Recurrence: T(n)

For some constant c,
T(n) ≤ 2 T(n/2) + *cn* when n > 2,
T(2) ≤ c

Unrolling Recurrence: 2 T(n/2) + cn

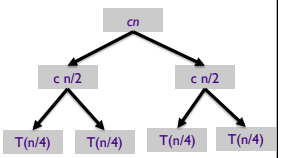
- First level: 2 T(n/2) + *cn*



How does the next level break down?

Unrolling Recurrence: 2 T(n/2) + cn

- Next level:



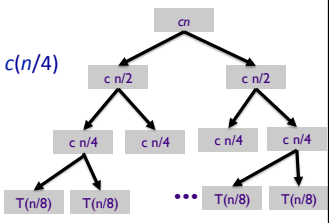
Each one is 2 T(n/4) + *c*(*n*/2)

Next level?

Unrolling Recurrence

- Next level:

Each one is 2 T(n/8) + *c*(*n*/4)

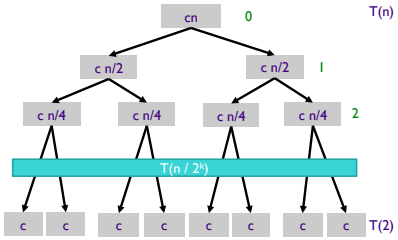


And so on…

What does the final level look like?

Unrolling Recurrence

- How much does each level cost, in terms of the *level*?
- How many levels are there (assuming *n* is a power of 2)?
- What is the total run time?



Unrolling Recurrence

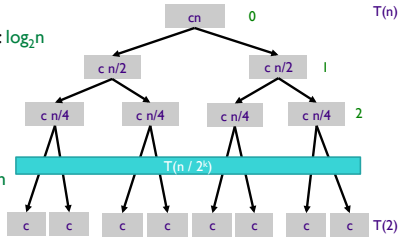
- How many levels are there (assuming *n* is a power of 2)?
- How much does each level cost, in terms of the level?
- What is the total run time?

Number of levels: $\log_2 n$

2^k problems
Size: $n/2^k$

Each level takes
$2^k * c * (n/2^k) = cn$

⇒ O(n log n)



Alternative: Proof by Induction

- Claim. If T(n) satisfies this recurrence, then T(n) = n $\log_2$ n.
  - Recall: T(n) = 2 T(n/2) + *cn*
- Pf. (by induction on n)
  - Base case: n = 2
  - Inductive hypothesis: T(n) ≤ cn $\log_2$ n
  - Goal: show that T(2n) ≤ 2cn $\log_2$ (2n)

Why doubling n?

Proof by Induction

- Claim. If T(n) satisfies this recurrence, then T(n) = n $\log_2$ n.
  - Recall: T(n) = 2 T(n/2) + *cn*
- Pf. (by induction on n)
  - Inductive hypothesis: T(n) ≤ cn $\log_2$ n
  - Goal: show that T(2n) ≤ 2cn $\log_2$ (2n)

$$
\begin{aligned}
T(2n) &\le 2T(n) + c2n \\
&\le 2cn \log_2 n + 2cn \quad \text{Replace w/ induction hypothesis} \\
&\le 2cn (\log_2(n) + 1) \\
&\le 2cn (\log_2(n) + \log 2) \\
&\le 2cn \log_2(2n) \checkmark
\end{aligned}
$$

Another Recurrence Relation: Binary Search

- How does binary search work?
- What is its recurrence relation?

Analyzing Binary Search

```
BinarySearch( L[1…n], key ):
     if len(L) == 1 and L[1] == key:
          return 1 #return the index
     else:
          return NOT_FOUND
     mid = n/2
     if L[mid] == key:
          return mid  #return the index
     if L[mid] < key:
          return BinarySearch(L[mid+1:], key)
     else:
          return BinarySearch(L[:mid], key)
```

What is the recurrence relation?

Analyzing Binary Search

```
BinarySearch( L[1…n], key ):
     if len(L) == 1 and L[1] == key:
          return 1 #return the index
     else:
          return NOT_FOUND
     mid = n/2
     if L[mid] == key:
          return mid  #return the index
     if L[mid] < key:
          return BinarySearch(L[mid+1:], key)
     else:
          return BinarySearch(L[:mid], key)
```

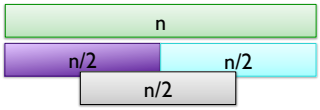
What is the recurrence relation?

T(n) = T(n/2) + c

Another Recurrence Relation

- Instead of recursively solving 2 problems, solve ***q*** problems
  - Size of problems is still n/2
- Combining solutions is still O(n)

Example: q=3:



What is the recurrence relation?

Another Recurrence Relation

- Instead of recursively solving 2 problems, solve ***q*** problems
  - Size of problems is still n/2
- Combining solutions is still O(n)
- Recurrence relation:
  - For some constant *c*,

    $T(n) \leq q\,T(n/2) + cn$ when $n > 2$

    $T(2) \leq c$

Intuition about running time?

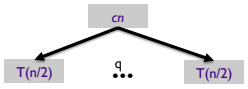
Unrolling Recurrence, q > 2

$T(n) \leq q\,T(n/2) + cn$

Unrolling Recurrence, q > 2

- First level:

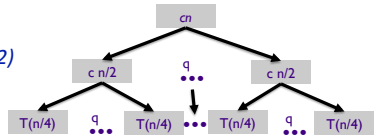
  $q\,T(n/2) + cn$



Unrolling Recurrence, q > 2

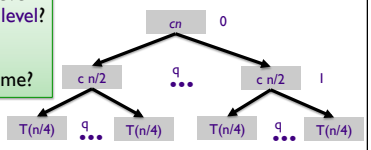
- Next level:

  $q\,T(n/4) + c(n/2)$



Unrolling Recurrence, q > 2

How much does each level cost, in terms of the level?
Number of levels?
What is the total run time?



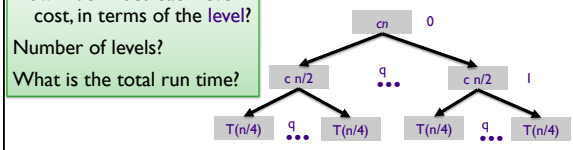
q^k problems at level k
Size: $n/2^k$

Number of levels: $\log_2 n$

Each level takes $q^k * c * (n/2^k) = (q/2)^i$ cn
→ Total work per level is *increasing* as level increases

Unrolling Recurrence, q > 2

How much does each level cost, in terms of the level?
Number of levels?
What is the total run time?



$T(n) \leq \Sigma_{j=0,\log n} (q/2)^i$ cn

Geometric series:
(constant ratio between successive terms)
Multiplying previous term by (q/2)

→ $O(n^{\log_2 q})$

Looking Ahead

- Problem 6 due Friday
- Exam 2 discussion…